A formal ORM-to -UML mapping algorithm

A Formal ORM-to -UML Mapping Algorithm

Peter Bollen

University of Maastricht, P.O. Box 616, p.bollen@mw.unimaas.nl, 6200 MD Maastricht, The Netherlands

Abstract

The object-role model (ORM) data structure can be represented in the unified modeling language (UML) using the five fact encoding constructs: class attribute, association, association class, sub-class and the association qualifier. In the existing literature there exist numerous mappings of how individual fact types from an ORM information model can be mapped onto 'well-formed' UML expressions. What is lacking in the existing literature is a precise description of the conditions on the 'source' object-role model under which a specific UML fact encoding construct can be applied in the 'target' UML class diagram. In this paper we will show under what conditions, a specific UML fact encoding construct must be applied in a way that results in a well-formed UML class diagram.

1 INTRODUCTION

The unified modeling language (UML) has become the de-facto standard for conceptualizing and specifying information systems, web-based applications and business- and software systems in general. Despite its upcoming inception as the world standard for expressing the results of the conceptualization and specification of the aforementioned types of systems, UML is considered to be : *'incomplete, inconsistent and unnecessarily complex'* [4]. This incompleteness, inconsistency and complexity, however, can be avoided when a conceptual schema design procedure (CSDP) from a fact-oriented modeling approach will be applied on *data use cases* [4]. The resulting conceptual schema will provide a 'semantic-rich' starting point for the creation of a UML class diagram. The fact-oriented approach that we will use in this article is *object role modeling* or *ORM* (see for a in-depth treatment of ORM [5]). The ORM approach and UML are also part of the enterprise features of *Microsoft's visual studio.Net* [6,7]. In [5] a detailed procedure (Rmap) for the mapping of a conceptual ORM schema onto a logical relational schema is given. Although several papers [4,5] show how individual ORM model fragments can be potentially encoded as fragments of UML models, a formal procedure that is similar to the Rmap procedure for mapping onto logical schemas [5, p.428] that specifies how a target UML class diagram can be created for any given 'source' ORM model is lacking.

The purpose of this paper is firstly, to analyze how the data structure in an Object Role Modeling (ORM) conceptual schema can be mapped onto UML modeling constructs, secondly, to find the properties of an ORM model that determine which fact-encoding construct in UML should be selected for the transformation of an individual ORM fact type, thirdly, to give a formal procedure that will derive a 'well-formed' UML class diagram for every 'well-formed' ORM data-structure and configuration of uniqueness and existence constraints.

2 ORM TO UML MAPPINGS FOR INDIVIDUAL FACT TYPES

2.1 Existence constraints in ORM and UML

In [5] it is stated that in ORM: " Each entity type in a completed conceptual schema plays at least one referential role and, unless declared independent…., at least one fact role. In general, the *population of an entity type equals the union of the population of its roles*. Unless the entity type is independent[1], its population is the union of the populations of its fact roles." [5, p.165]. This means that the modeling default in ORM is the situation in which an entity type is *not* declared independent, and therefore, instances or entities of an entity type are *not* allowed to exist on their own.

In UML *application concepts* or *entity types* must be modeled as *object classes*. Contrary to the convention in ORM in which instances of entity types by default have to play *at least* one role of all the roles they are involved in, instances of UML object classes are allowed to exist on their own, by default.

[1] Which is shown by adding an exclamation mark (!) next to its name.

For each entity type that has been encoded in UML as an *(explicit) object class* a *textual constraint* needs to be added to the *object class* and the *associations* or *attributes* that are connected to that object class that specifies that instances of that object class are *not* allowed to exist on their own (see figure 1). In case an ORM entity type is declared independent such a textual constraint is not needed in UML. This means that for an ORM-to-UML transformation we need *global* knowledge of the input ORM conceptual schema (the independency status of an object type) in order to transform ORM object types onto UML object classes (see figure 1).

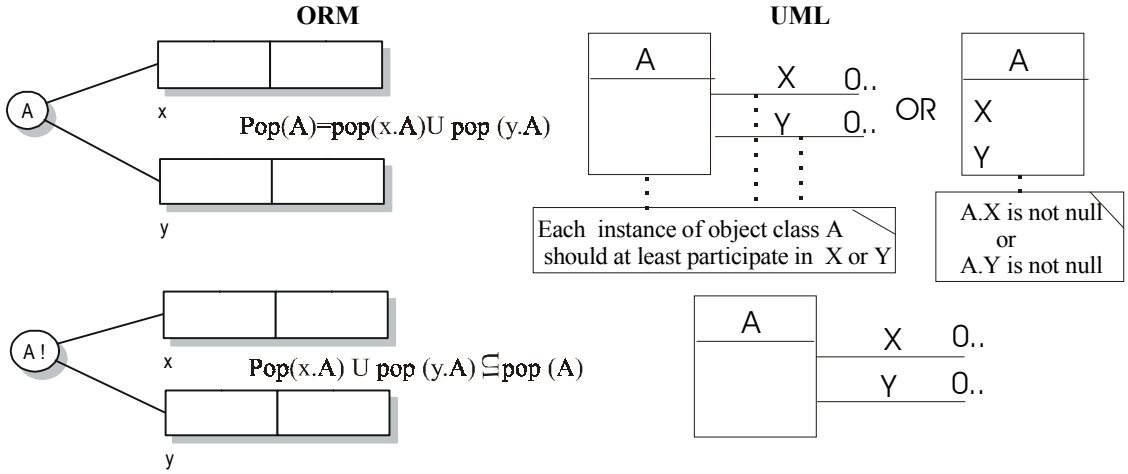


Fig. 1: Dependent versus independent entity types

If for at least one role that is played by an ORM object type a mandatory role constraint is defined then there will be no textual constraint in the UML model, but a corresponding lower multiplicity of *1..* or a textual constraint (see figure 2) attached to the applicable *association end* or *class attribute* .

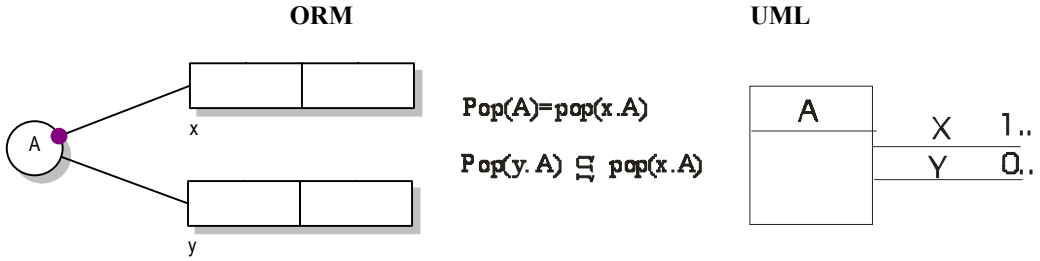


Fig. 2: Mandatory role in ORM and the mapping to UML

2.2 ORM object types modeled as object classes and attribute types in combination

In this section we will consider the conditions under which an ORM object type can be encoded as an object class and/or as an attribute type. When an object type is encoded as an attribute type exclusively, there will be no object class or data type symbol in the UML class diagram. The defining UML literature states that the *type of an attribute*: " designates the classifier whose instances are values of the attribute. Must be a Class, Interface or Data Type." [8, UML semantics guide v.1.3, p.2-26] and with regard to *attributes*: " are generally used for pure data values without identity." [9, p. 42]. This means that if we encode an ORM entity type as the attribute type and the role it plays in a binary fact type as the attribute it is not possible to connect the data values for the attribute with the object instances of the object class for the entity type because: " A data value is a member of a mathematical domain – a pure value. Two data values with the same representation are indistinguishable; data values have no identity." [9,p.248]. This means that the encoding of an ORM entity type as an UML attribute (type) in the integrated UML class diagram is

only allowed if instances of this entity type are *not* allowed to exist independently and no mandatory roles [2] are defined on this entity type.

2.3 Unary fact type configurations

In [2] two ways of encoding unary fact types are given. A unary fact type can be encoded as a *boolean* attribute or as a *sub-type*[3]. In figure 3a we have given the first example configuration for a unary in ORM.

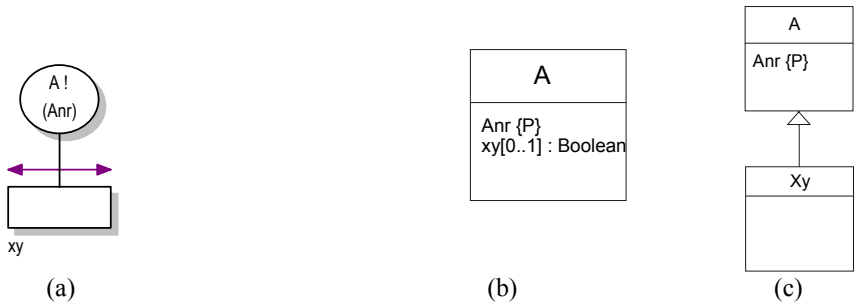


Fig.3: (a) Example 1.1 and (b) first encoding in UML and (c) second encoding in UML

Example 1.1 can be encoded as a *boolean* class attribute in UML (see figure 3b) to which we have assigned an explicit attribute multiplicity of [0..1] that implies that instances of object class A can exist independently. The second option that exists for the encoding of a unary (ORM) fact type (see figure 3c) is the *subtype* fact-encoding construct in UML [5]. We note that the default attribute multiplicity is [1..1] in those cases in which no multiplicity is specified [9, p.169].

2.4 The application of the attribute fact encoding construct for binary fact type configurations

In this section we will summarize the ORM to UML (fact) mappings that we have encountered in the literature [4-5] for *binary* ORM fact types in which we want to apply the class attribute fact encoding construct in UML. We note that all mappings in this section can only be applied for binary fact types in which A and B are *distinct* entity types.

2.4.1 M:N cases of binary fact types

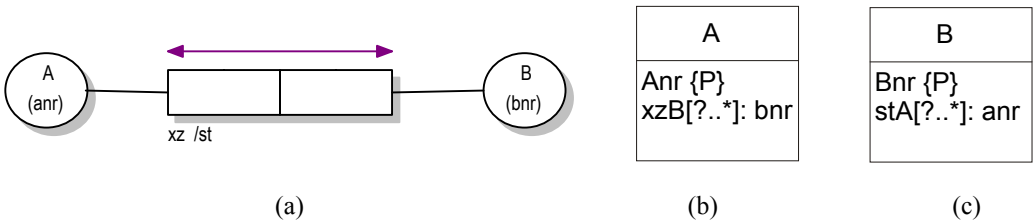


Fig. 4: (a) Example 2.1 and (b) first encoding in UML and (c) second encoding in UML

Example 2.1 in figure 4a is a binary fact type that has a uniqueness constraint that spans both fact roles. This fact type is encoded in UML either as the attribute *xzB* in figure 4b *or* as the attribute *stA* in figure 4c. The uniqueness constraint is mapped as the maximum attribute multiplicity of *..*]* in the UML class diagrams of figures 4b and 4c. Furthermore we have combined the predicate *xz* and the entity type *B* in figure 4b and the predicate *st* and the entity type *A* in figure 4c as one expression, namely the attribute name

[2] In the sense of non-implied mandatory role constraints including non-implied disjunctive mandatory role constraints

[3] In [4] it is given that a binary association can replace a unary. We do not consider this as an option for the encoding of a *unary* fact type but rather a transformation in the ORM domain in which a *unary* fact type is transformed into a *binary* fact type. The ORM to UML transformations for binary fact configurations is subject of section 2.4.

of the UML class attribute. The reference types (anr or bnr) are modeled as the attribute type. From the point of view of an individual fact type the choice between both UML options is arbitrary. In addition to the individual fact type semantics we need to consider the semantics of the integrated UoD as well. In the ORM conceptual schema in 4a it is modeled that instances of entity types A and B are not independent. However, if we consider the example fact type without global knowledge of the UoD we can not determine whether the roles that are played by these entity types in this fact type are *implied* mandatory roles or not. For now we will assign a question mark (?) to the lower attribute multiplicity that indicates that the precise value for the lower attribute multiplicity can only be determined when we have knowledge of the integrated ORM conceptual schema.

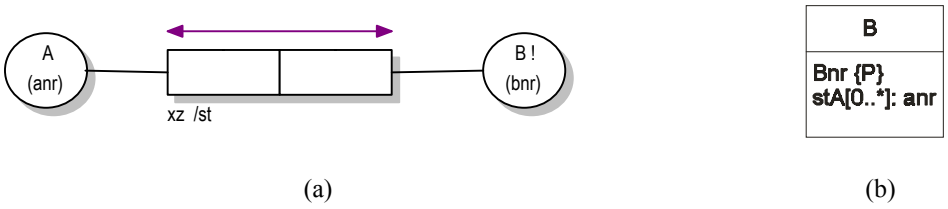


Fig. 5: (a) Example 2.2 and (b) encoding in UML of example 2.2

In the example 2.2 in figure 5a we have explicitly shown that instances of entity type B are allowed to exist independently. In the corresponding UML class diagram (figure 5b) this is shown by overruling the default ([1..1]) attribute multiplicity of object class B and assigning a lower attribute multiplicity of *[0..* . Furthermore the presence of a uniqueness constraint defined on both roles implies the maximum attribute multiplicity of *..*]* in the UML class diagram of figure 5b.

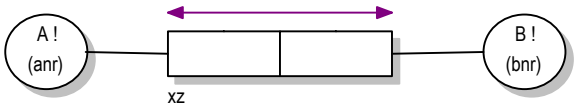


Fig. 6: Example 2.3

In the binary fact type configuration of example 2.3 in figure 6 we have two entity types that are *independent*. Applying the attribute construct here will lead to the violation of the first condition that was given in section 2.2. In these situations we can *not* apply the attribute fact encoding construct for binary fact type configurations. This fact type configuration should be modeled as an association (see section 2.5) or as an association class (see section 2.7).

2.4.2 1:M cases of binary fact types

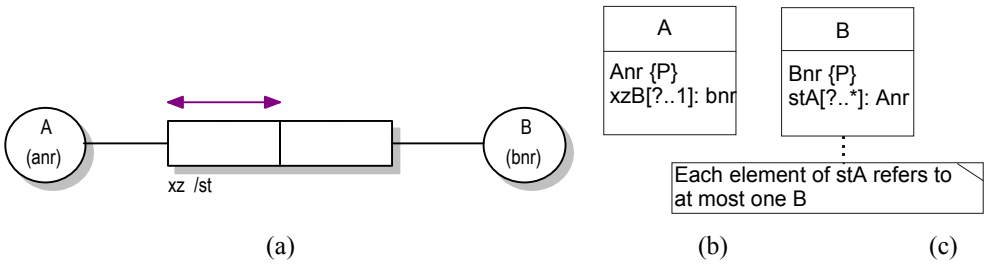
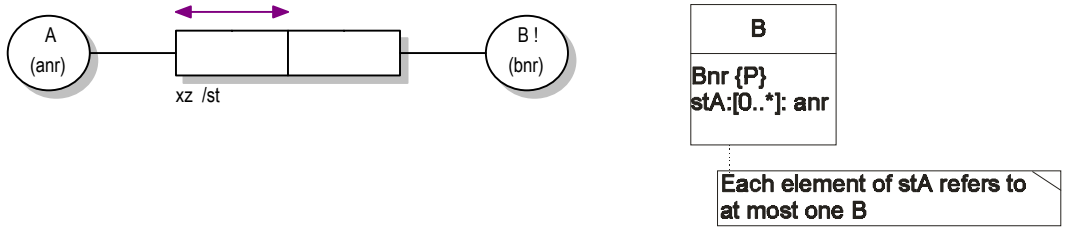


Fig. 7: (a) Example 2.4 and (b) first encoding in UML (c) second encoding in UML

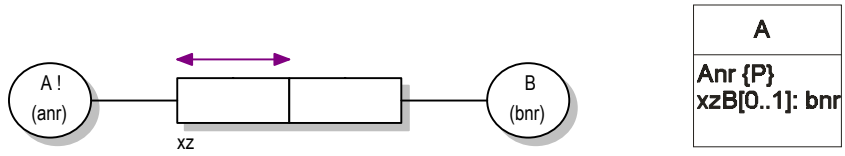
The textual UML constraint: {Each element of stA refers to at most one B} is the encoding in UML of the uniqueness constraint that is defined on the left-hand role in example 2.4. We note that without additional knowledge of the integrated ORM conceptual schema, the choice between the first option (figure 7b) and the second option (figure 7c) in UML is arbitrary.



(a) (b)

Fig. 8: (a) Example 2.5 and (b) encoding in UML of example 2.5

In example 2.5 the uniqueness constraint is defined on the role that is played by entity type A. Since entity type B is independent we have to 'center' the UML encoding of the fact type with predicate *xz/st* around object class B in figure 8b. In UML we cannot capture all semantics of example 2.5 using the standard notation. We have to encode the uniqueness constraint that is defined on the role that is played by entity type A as a textual constraint defined on object class B [5].



(a) (b)

Fig. 9: (a) Example 2.6 and (b) encoding in UML of example 2.6

In example 2.6 we note that instances of entity type *A* are allowed to exist on their own. This will be encoded in UML by adding a lower multiplicity of *0..* to the *xzB* attribute. The uniqueness constraint that is defined on the left role in example 2.6 in figure 9a will be encoded a maximum multiplicity of *..1]* for the *xzB* attribute of object class A in figure 9b.

2.4.3 1:1 cases of binary fact types

In figure 10 we have given the remaining binary fact type configurations that can be mapped onto UML class attributes.

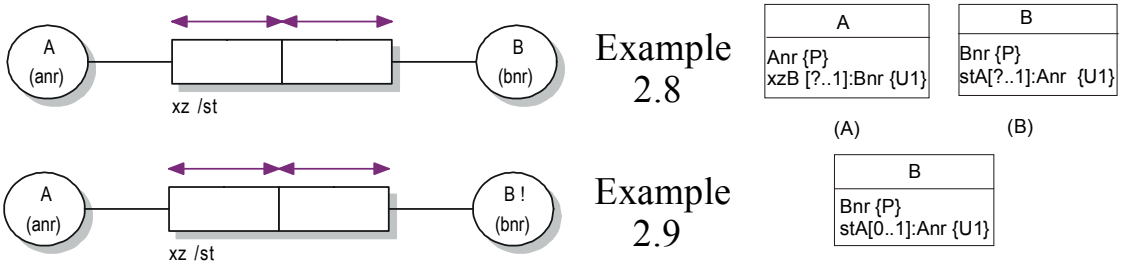


Fig. 10: Examples 2.8 and 2.9 and the encoding in UML

We note that the uniqueness constraint that is defined on the right hand role of the ORM example 2.8 is encoded as the attribute uniqueness constraint U1 (see [5, p.355]) in the UML modeling option A.

2.5. The application of the association construct for binary fact types

The three binary ORM fact configurations in examples 2.3, 2.7 and 2.10 in figure 11 can not be encoded as class attributes. The other 7 configurations can in principle be encoded as associations as well as attributes in UML. This means that choices exist regarding the encoding of an instance of a binary fact

configuration in an ORM model onto a UML class diagram. In [3,5; p. 365-366] it is illustrated for all binary fact configurations how they map onto the UML association construct. If instances of an *entity type* or *nested object type* are independent then the corresponding lower association end multiplicity on the 'far' end is *0*... In case of a mandatory role, the lower association end on the 'far' end is *1*.. . In many cases the knowledge of the individual fact type configuration is not sufficient to establish a complete ORM-to-UML transformation. As we noted in section 2.1, knowledge of the integrated ORM conceptual schema is needed in order to decide whether additional textual participations constraints must be added to the associations and attributes in the UML class diagram.

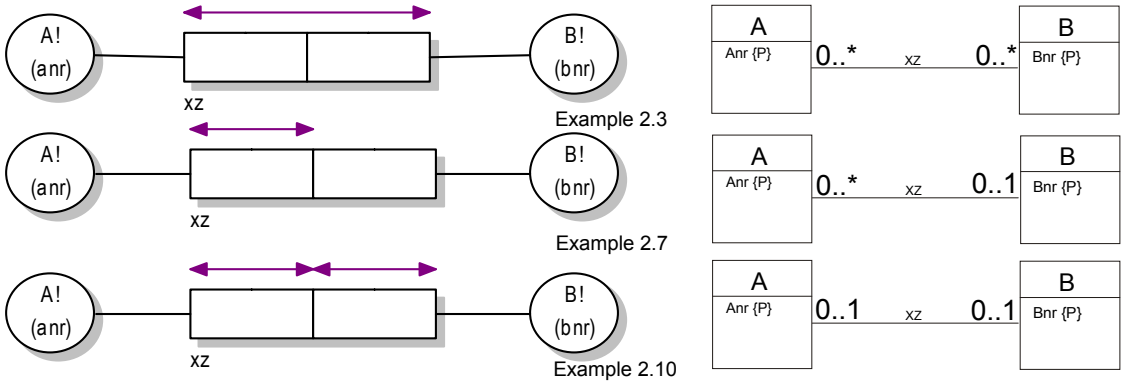
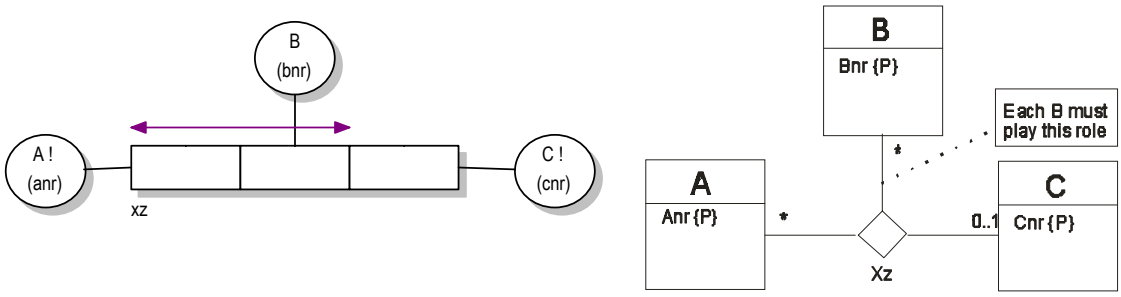


Fig. 11: Remaining binary ORM fact configurations and their mapping onto UML associations[4]

If a uniqueness constraint is defined on exactly one role in a binary fact type then this will map onto an maximum association end multiplicity of *..1* on the 'far' association end. The maximum association end multiplicity of ..* in a binary fact type points at the absence of a uniqueness constraint defined on the 'other' role.

2.6 N-ary fact type configurations

N-ary (N>2) fact types in ORM can be encoded in UML using either the *association* construct or the *association class* construct. In [2,4,5] examples can be found that illustrate how ternary ORM fact types can be mapped onto (ternary) associations in UML. The modeling semantics for ternary fact types and associations can be generalized to any N-ary relationship [5].



(a) (b)

Fig. 12: (a) ternary ORM fact type and (b) the UML mapping

A uniqueness constraint that is defined on the roles played by entity types A and B in figure 12a is

[4] It should be noted that in UML two short hand notations exist for lower, maximum multiplicity combinations. If * is the single association end multiplicity that is specified it means: *0..**. If *1* is the single association end multiplicity that is specified it means: *1..1* [8].

transformed as the maximum association end multiplicity (AEM) of *1* for the opposite role played by object class C in the corresponding UML association in figure 12b. In [4] it is argued that "….there are many cases with n-ary associations where UML's multiplicity notation is incapable of capturing even a simple mandatory role constraint." [4, p.6] proposes to use a textual constraint in UML for encoding a(n) (implied) mandatory role while maintaining the minimum AEM of 0 (see figure 12b).

2.7 Nested object types

In addition to the *simple (abbreviated) reference scheme*, ORM has the concept of *nested object types*. Compound reference schemes that originate in user verbalizations of data use cases will result in nested object types (see figure 13).

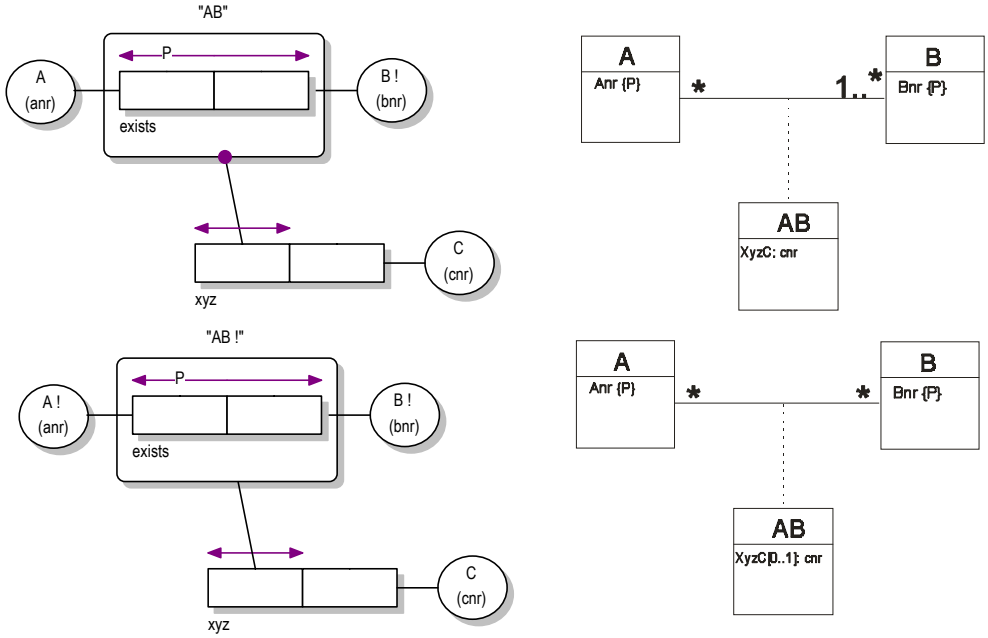


Fig. 13: Mapping ORM nesting in functional fact type to UML association class

The UML association class can be applied for encoding *nested object types* that are involved in *functional* fact types [9], in that case the fact type is encoded as the class attribute (see figure 13).

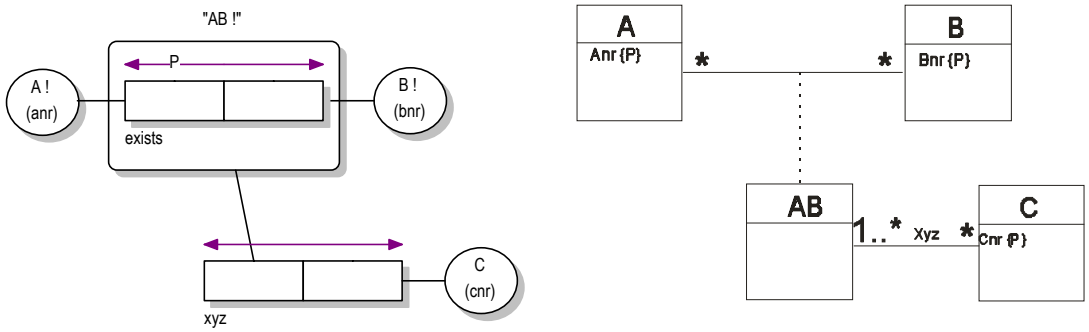


Fig. 14: Mapping ORM nesting in non-functional fact type to UML association class

The association class can also be used for the encoding of a nested object type and a *non-functional* fact type in that case the association class will participate in a binary association that is connected to an object class that is the encoding of the entity type in the other role (see figure 14).

In the case of atomic entities there exist choices for the encoding of some binary fact type configuration either as an *attribute* or an *association*. In the case of *nested object types* these choices are limited because all non-functional fact types that involve a nested object type have to be encoded in a separate association. However, the existing ORM-to-UML literature does not explicitly cover the case in which the entity type that plays the functional role in the fact type is *independent* (entity type C in figure 15). We have shown in section 2.2 that such an entity type has to be encoded as an *explicit* object class.

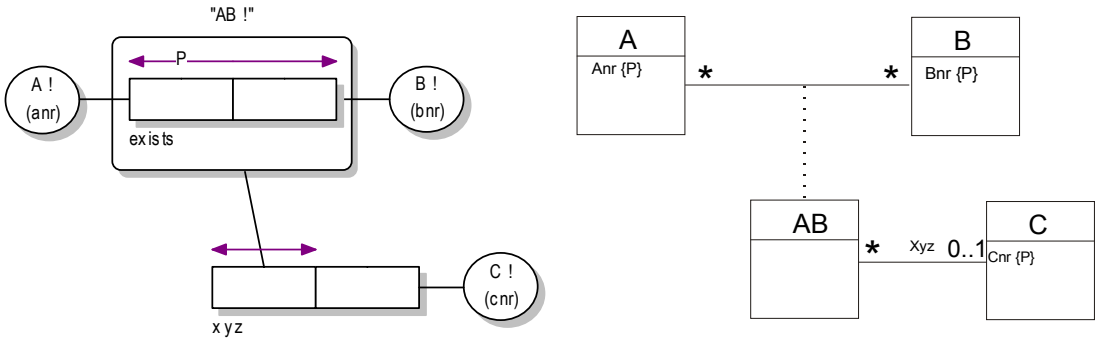
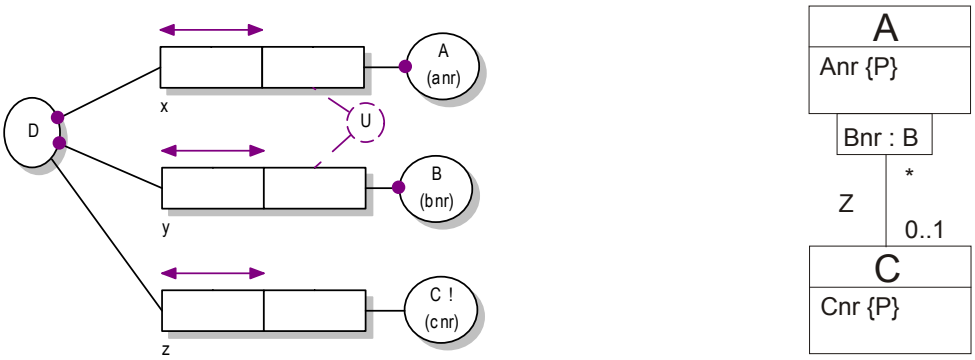


Fig. 15: Mapping ORM nesting in functional fact type to UML association class and association

The mapping of N-ary associations onto the *association class* construct can be redefined as the mapping of a binary association in which at least one of the participating object types is a *nested* object type. The decision whether to map such a fact type onto an association class attribute and its accompanying attribute multiplicity *or* an additional binary fact type is in essence identical to the decisions that should be made in case of a binary fact type with only atomic entity types.

2.8. Co-referencing

The third way of encoding reference schemes in ORM is called *co-referencing* [5]. The reference types in the co-reference scheme are characterized by a external uniqueness constraint that is defined on the roles in which they participate. The modeling construct in UML that can be used to encode co-referencing is the *association qualifier* (see figure 16b).



(a) (b)

Fig. 16: ORM-to-UML mapping (a) co-reference scheme onto (b) association qualifier.

In [5] it is noted that the UML mapping of the ORM input model in figure 16 is incomplete because the entity type that is identified in the co-reference scheme has disappeared in the resulting UML class diagram. In [5, p.396] it is stated that: " some cases of co-reference could be mapped into qualified associations, but mapping to separate attributes or associations supplemented by a textual composite

uniqueness constraint offers a more general solution." In figure 17 we have given this more general solution in which we basically follow the ORM-to-UML mapping logic of a *simple* (abbreviated) reference scheme in which the reference types are encoded as class attributes while adding the {P} qualification to those attributes. We note that if at least one of the object types that plays the functional role of the co-referencing fact types is independent we are not allowed to encode these fact type(s) as (a) name attribute(s) for reasons previously mentioned. The overall transformation algorithm in appendix C will check these conditions in order to map a fact type within a co-reference scheme as an *identifier attribute* or to encode the *compound* reference scheme as an association class[5].

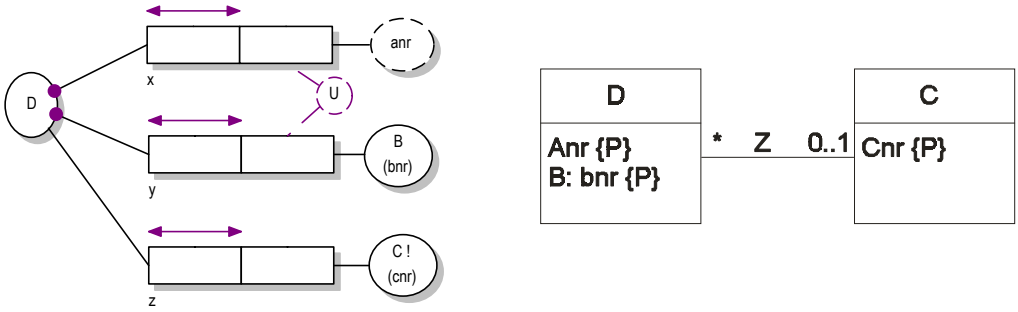
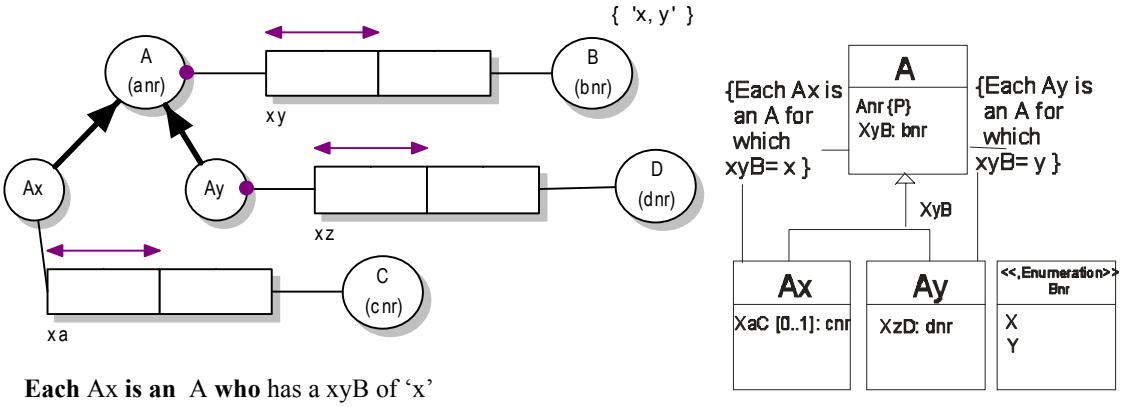


Fig. 17: ORM-to-UML mapping (a) co-reference scheme onto (b) class attributes

2.9 Sub typing

The concept of subtype in ORM corresponds to the concept of *subclass* in UML. In [5] it is noted that in UML formal subclass definitions need not to be recorded for subclasses. Halpin [5] proposes to add an OCL or textual constraint in the corresponding UML model.



Each Ax **is an** A **who** has a xyB of 'x'
Each Ay **is an** A **who** has a xyB of 'y'

Fig.18: ORM-to-UML sub type mapping

3 THE OVERALL ORM-TO-UML MAPPING

In this section we will determine under what conditions on the 'overall' ORM conceptual schema we can use a specific modeling option in UML to encode a 'specific' ORM fact type. The properties of ORM object types will lead to a restriction of the choices for the *fact-encoding* construct as soon as we have knowledge on the final ORM model of an application area. In section 2.2 we concluded that an *object type*

[5] In which case we implicitly apply the nest/coreference theorem in [5, p. 594-595]

is either modeled an object class or as an attribute type. This leads to rules 1 and 2:

- Rule 1: An entity type is modeled as one or more attribute (types) OR
  an entity type is modeled as an object class.
- Rule 2: IF a value constraint is defined on the value type
  THEN model this value type as an *enumeration data type*
  eventually in combination with an implicit data type as
  attribute type of one or more attributes
  OR model this value type as the *attribute name* of the identifier attribute of an
  entity type that is modeled as an *object class*.
  ELSE model this value type as an *implicit data type* as attribute
  type of one or more *attributes* OR model this value type as the *attribute name* of
  the identifier attribute of an entity type that is modeled as an *object class*.

We have discovered in section 2.1. that if an object type is *not* independent but it is involved in at least *one* mandatory role then this object type can *not* be modeled as an attribute (type) in UML.

- Rule 3: IF an entity type is independent AND has an abbreviated reference scheme
  THEN model this entity type as an *explicit object class* and model the value type of the
  reference scheme as the attribute name of the identifier attribute
- Rule 4: IF an entity type plays two or more roles AND is involved in at least one mandatory role
  THEN model this object type as an object class.

If one or more participating fact types in the co-reference cannot be modeled as (a) class attribute(s) then the entity type that is co-referenced has to be encoded as a *nested object type*.

- Rule 5: IF an entity type has no abbreviated reference scheme
  THEN IF it is co-referenced
  THEN IF a constituting object type exists that must be encoded as an object
  class
  THEN transform the co-referenced entity type into
  a nested object type
  ELSE Create an object class for the co-referenced entity type and a
  name attribute for each constituting fact type of the co-reference.

If an ORM entity type is involved in a sub-type relationship either as *subtype* or as *super type* it has to be encoded as an *explicit object class* in the UML model. This results in rule 6:

- Rule 6: IF an ORM entity type is a subtype and/or super type
  THEN model this object type as an object class

Furthermore, we know that when an entity type plays at least one role in a *N-ary* fact type (N>2) or at least one role in a unary fact type then this entity type *must* be encoded as an object class.

- Rule 7: IF an entity type plays at least one role in a N-ary fact type
  OR at least one role in a unary fact type
  THEN model this entity type as an object class

In section 2 we concluded that we can apply the attribute fact encoding construct in UML for the encoding of 7 out of 10 binary fact configurations under the condition that the entity types *A* and *B* are *distinct*. This leads to the following rule:

- Rule 8: IF an entity type plays two roles in the same binary fact type at least one time
  THEN model this entity type as an object class

For those binary fact types in which the role that is played by a nested object type is encoded as an association class, additional constraints exist with respect to the choice of attribute or association as a fact encoding constructs.

- Rule 9: IF a nested object type in an binary fact type is encoded as an association class
  in UML AND a uniqueness constraint is defined on both roles in a binary fact type
  THEN the other entity type in that binary fact type must be encoded as an object class.

This leaves us now with those entity types that are involved in at least one binary relationship with another entity type that is not yet encoded as an object class and that do **not** play 'global' mandatory roles. This requires an optimization as to decide which of the object type(s) has to become the object class around which the other object types and fact type predicates can be centred as attributes. We will call this remaining group of entity types REST. We will now minimize the total number of object classes.

- Rule 10: List the members of REST in the order of the number of roles that is played by each entity type.
  Take the highest ranked entity type (that is the entity type that is involved in the highest number of fact types) and create an object class for this entity type. The entity types that participate in the other roles of these three fact types will be encoded as a class. Adapt REST and the list will be scanned for the next entity type having the highest number of roles in which it participates.

The algorithms in which these ORM-to-UML mapping rules are embedded can be found in appendix C

3.1 An example of the ORM-to-UML tranformation

In this section we will illustrate how the ORM-to-UML mapping procedure will be applied in practice.

3.1.1 The first part of the transformation.

The input ORM example for the transformation is given in figure 19. The algorithm for the first part of this transformation is algorithm 9a in section C.2 of appendix C.

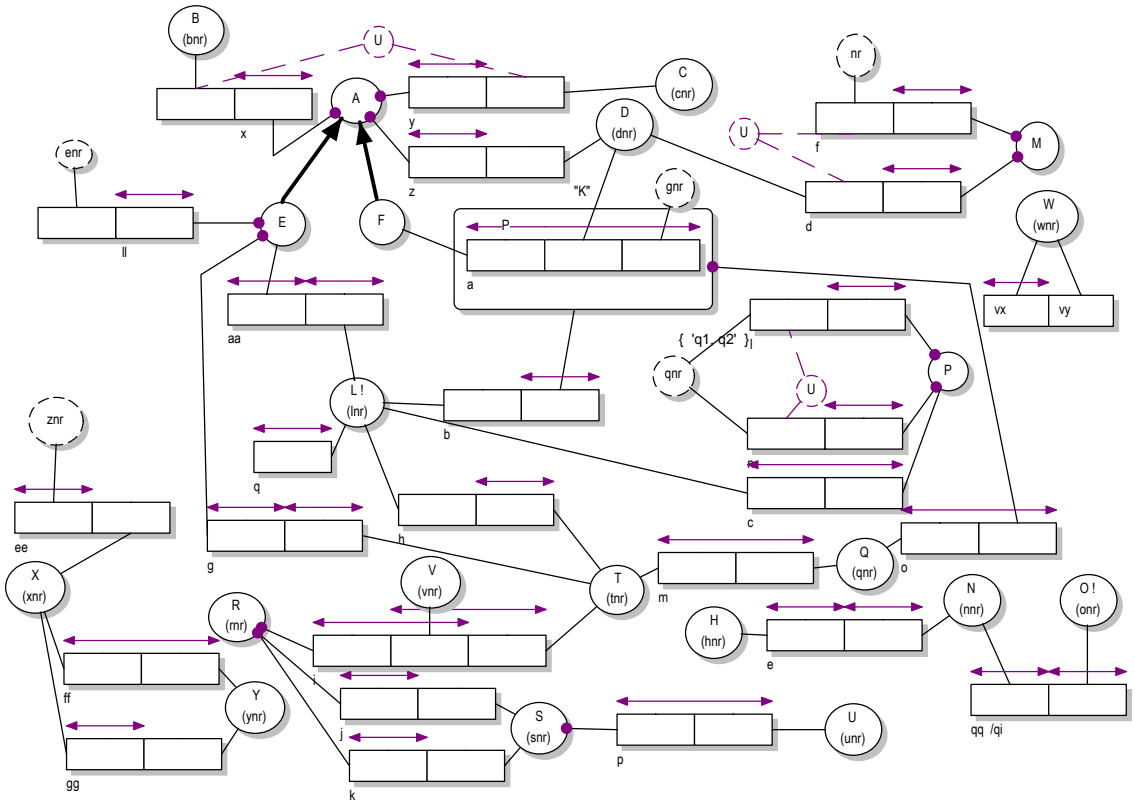


Fig.19: Input ORM example

In figure 20 in we have created the explicit object classes, that follow from the application of rules 1, 3, 4, 6, 7 and 8. Furthermore, we have created the enumeration data types and the unary, ternary and higher order fact types and the binary fact types in which both roles are played by the *same* object type We note that the co-referenced entity type A that is also a super type can still be encoded as an *object class* or as an *association class*. Furthermore the constituting object types B and C of the co-referenced entity type A in

principle can be encoded as attribute type of an object class A *or* as constituting classes of an association class A.

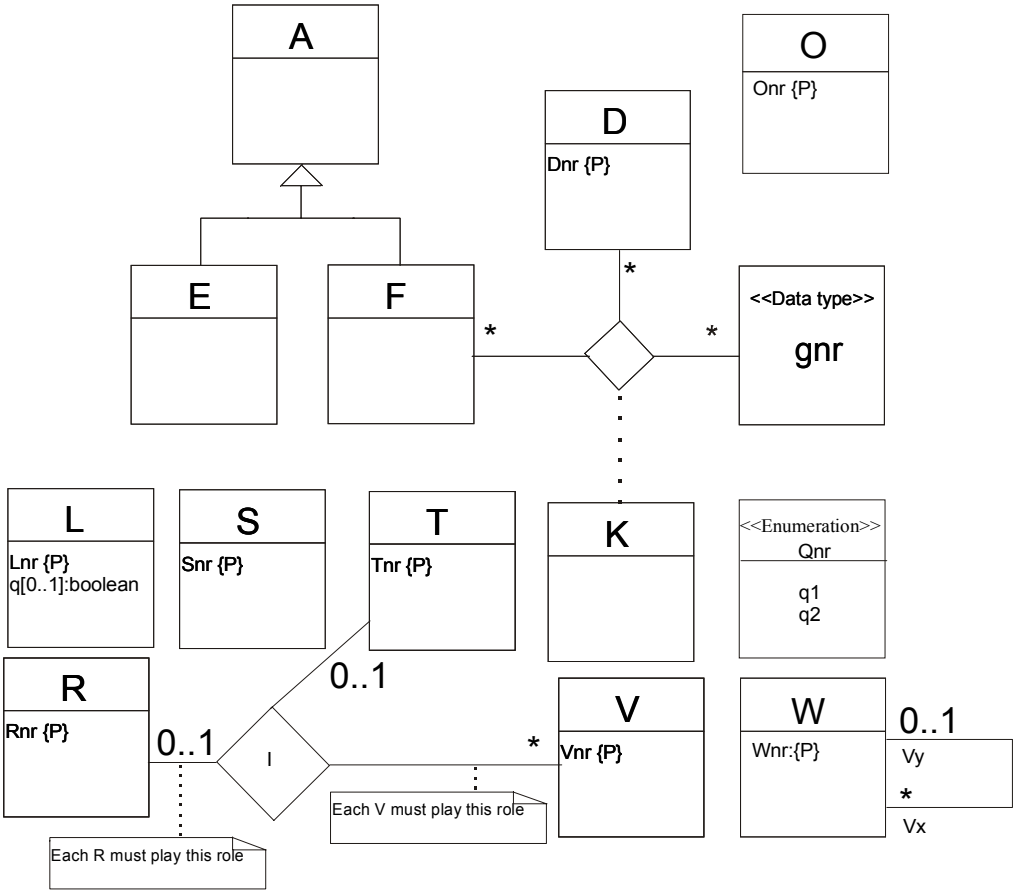


Fig.20: 'initial UML class diagram' for the ORM conceptual schema in figure 19

3.2 The second part of the transformation

The second part of the transformation will map the *non-independent* entity types that play exactly one role in a binary fact type and that can be attached to the object class of the other entity type of the fact type if the other entity type is encoded as an object class.

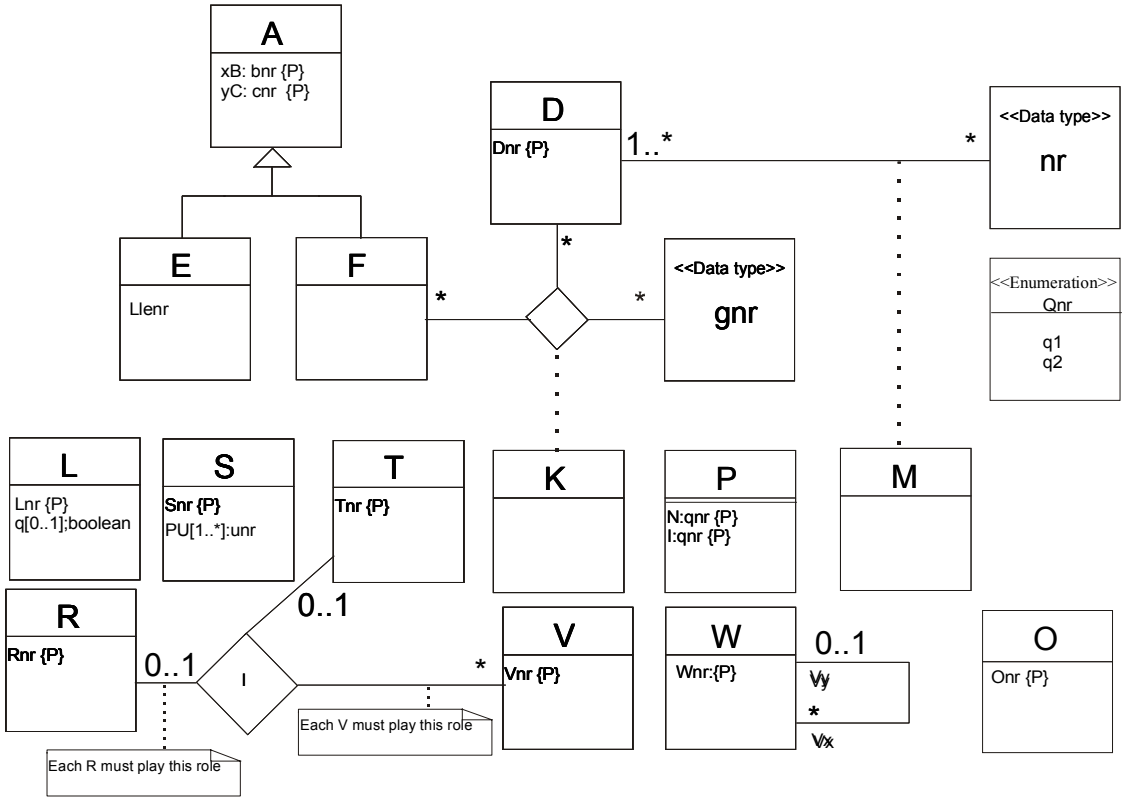


Fig. 21: 'second UML class diagram' for the ORM conceptual schema

For example fact type *p* and entity type *U* are encoded as attribute *pU* of object class *S*. The abbreviated reference scheme *unr* can be encoded as the attribute type *unr* of attribute *pU* of object class *S* in figure 21. Furthermore, entity types *B* and *C* and their respective co-reference fact types *x* and *y* can be mapped onto the class attributes x*B* and *yC* of object class *A*. Co-referenced object type M from the ORM example in figure 19 cannot be encoded as an explicit object class because one of its constituting object types (D) is involved in a ternary fact type. This leaves only the modeling option of association class (see figure 21), which has the *D* and *nr* as constituting classifiers. Co-referenced object type *P* can be encoded as a simple object class having two name attributes *n* and *i* and accompanying {P} constraints. Finally, the functional value type *enr* is encoded as attribute of object class *E*. The application of algorithm 9b in appendix C will lead to the second UML class diagram in figure 21.

3.3 The third part of the transformation

The next group of entity types that will be mapped is the group of entity types that play roles in two or more binary relationships with (different) ORM entity types of which at least one is encoded as object class(es) or association class(es). An example of such an entity type is entity type *Q* in figure 19. This entity type must be encoded as an explicit object class and hence the association *o* must be encoded as a binary association between the association class *K* and object class *Q* (see figure 23). Another example of such an entity type is entity type *N*. Entity type *N* and fact type *qq* can be encoded as class attribute *qi* of object class *O* in figure 22. Consequently entity type *H* in figure 19 will be encoded as object class *H* in figure 22 and fact type *e* will be encoded together with entity type N as class attribute *eN* of object class *H*. The result of algorithm 9c in appendix C in which rules 1,2 and 9 are embedded results in the third UML class diagram in figure 22.

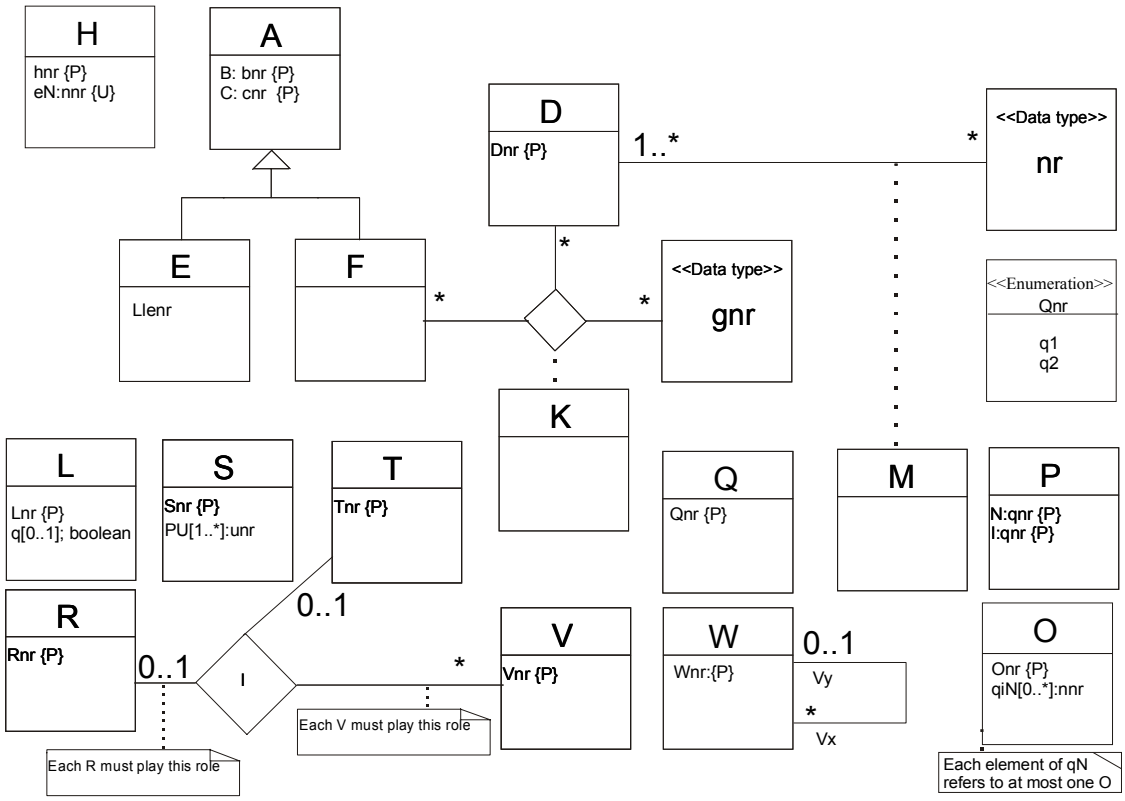


Fig. 22: 'third UML class diagram' for the ORM conceptual schema

3.4. The fourth part of the transformation algorithm

This leaves us now with those entity types that are involved in at least one binary relationship with another entity type that is not yet encoded as an object class and that do **not** play 'global' mandatory roles. This requires an optimization in the fourth part of the transformation algorithm as to decide which of the object type(s) has to become the object class around which the other object types and fact type predicates can be centred as attributes. We will call this remaining group of entity types REST. In the example from figure 16, REST consists of the object types *X, Y, znr*. We will now optimize the output UML class diagram by minimizing the total number of object classes. We will list the members of REST in the order of the number of roles that is played by each entity type. The algorithm will take the highest ranked entity type, that is the entity type that is involved in the highest number of fact types. In this example this will be entity type X (3 times). An explicit object class will be created for this entity type. The entity types that participate in the other roles of these three fact types will be encoded as class attributes and will therefore be taken out of the ranking list, since they can only be encoded as attribute (types). Subsequently, the list will be scanned for the next entity type. The result of the application of algorithm 9d in appendix C can be found in figure 23.

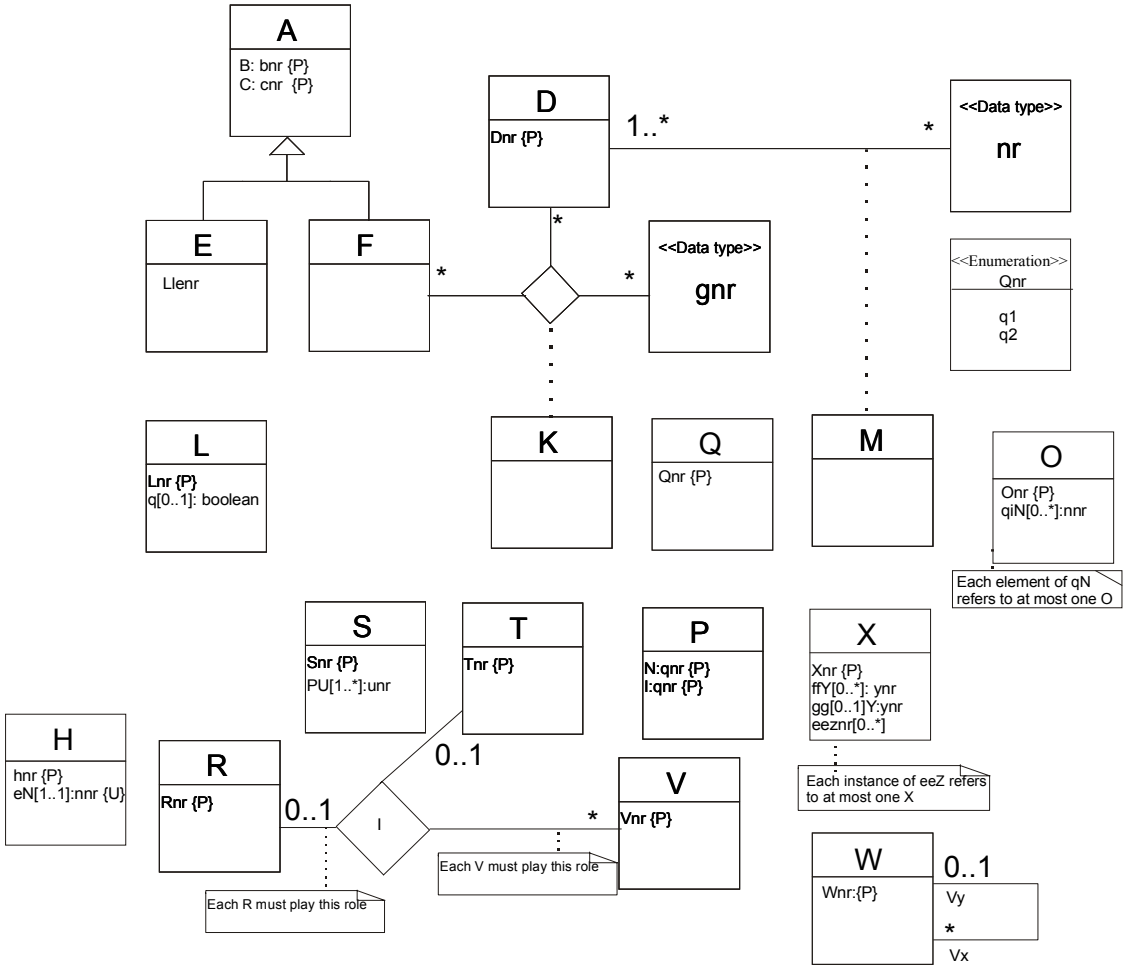


Fig. 23: 'fourth UML class diagram' for the ORM conceptual schema

3.5. The final part of the transformation.

The final part of the 'integrated' ORM-to-UML mapping will take care of the mapping of the binary fact types that not yet have been encoded onto associations and their corresponding association end multiplicities. Furthermore, in this last step the global textual participation constraints will be added to encode the ORM default mandatory participation of entities in at least one of the fact types in which they participate. The result of applying algorithm 9e in appendix C and conclusively the final 'well-formed' UML class diagram that expresses all the semantics of the input ORM model is given in figure 24.

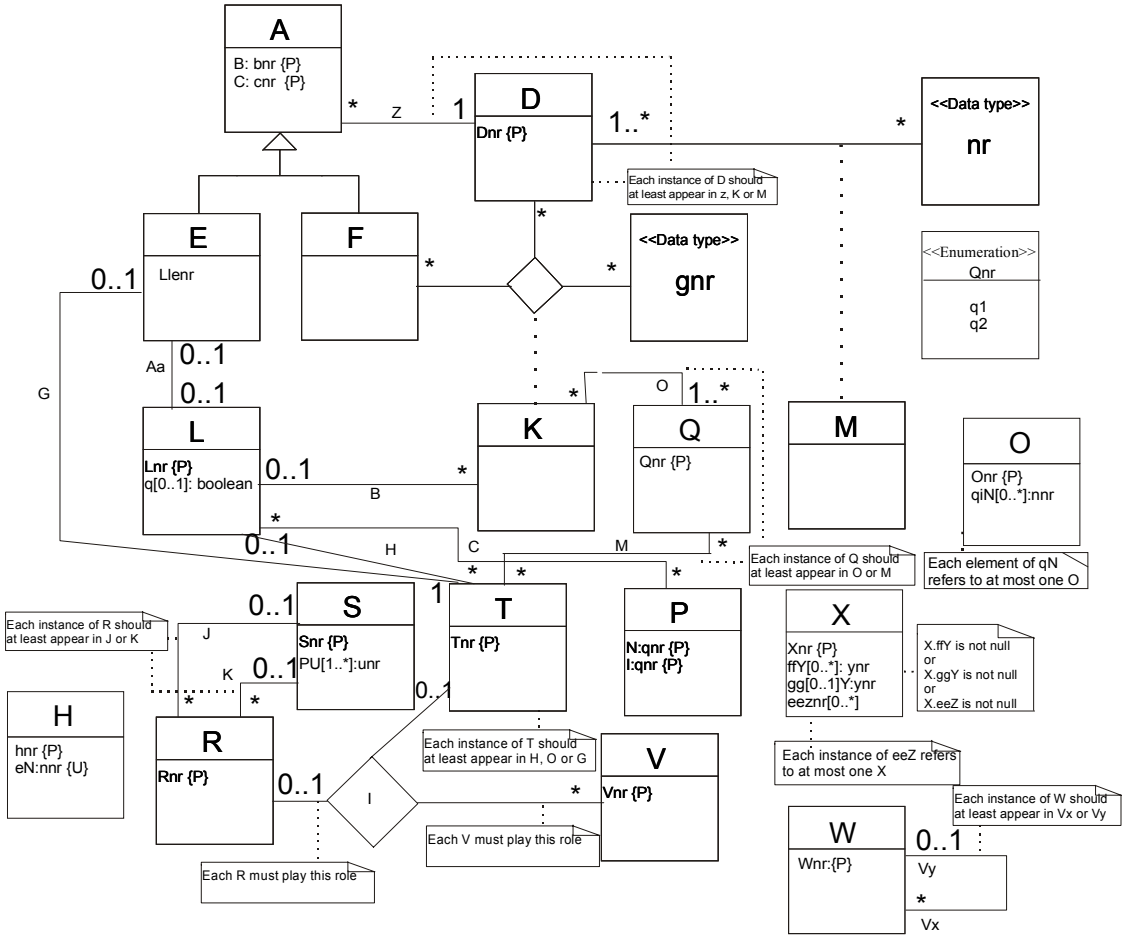


Fig. 24: final 'UML class diagram' for the input ORM conceptual schema in figure 19

4 CONCLUSION

In section 2 of this paper we have analyzed the UML modeling constructs that can be used for the encoding of reference schemes, fact types and mandatory role, global participation-, value- and uniqueness constraints in ORM. The five fact encoding constructs in UML are intertwined with the modeling constructs for population constraints. This means that in order to encode the UML data structure we need to have knowledge of existence constraints and uniqueness constraints or the integrated ORM conceptual schema. In section 3 we haven given formal rules that constrain the allowed UML class diagrams that can be created out of an ORM conceptual schema. These rules can be implemented in an ORM-to-UML mapping procedure, comparable to the Rmap procedure for the conceptual to logical transformation. The general ORM-to-UML mapping algorithm for this procedure is given in appendix C. In section 3.1 we have illustrated the application of this formal ORM-to-UML mapping algorithm on a significant (generalized) example of an application ORM conceptual schema.

REFERENCES

[1] G.Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, Reading MA, USA, 1999.

[2] T. Halpin, Data modeling in UML and ORM revisited, in: Proceedings EMMSAD'99, 1999.

[3] T. Halpin, Data modeling in UML and ORM: a comparison, Journal of Database Management, 10, 1999.
[4] T. Halpin, Augmenting UML with Fact-orientation, in:workshop proceedings: UML: a critical evaluation and suggested future, HICCS-34 conference, 2001.
[5] T. Halpin, Information Modeling and Relational Databases, Morgan Kaufmann Publishers , 2001.
[6] T. Halpin, Microsoft's new database modelling tool: Part 1, Journal of Conceptual Modeling, June 2001
[7] Microsoft visual studio.net, http://msdn.microsoft.com/vstudio/nextgen/whatsnew.asp, july 2001
[8] OMG, UML Specification v. 1.3 final draft, OMG UML Revision Task Force website, http://www.omg.org/cgi-bin/doc?formal/2000-03-01accessed 30 august, 2001
[9] J.Rumbaugh, J. Jacobson, G.Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, Reading MA, USA, 1999.

APPENDIX A: ORM MODELING CONCEPTS

Object-Role Modeling (ORM) is a methodology for modeling information systems on the conceptual level. It is named after its main constituents: *objects* that play *roles* in relationships. The 'role-based' ORM notation makes it easy to define *static* constraints on the data structure and it enables the modeler to populate ORM schemas with example sentence instances for constraint validation purposes. In ORM (and other fact oriented approaches) the *fact* construct is used for encoding *all* semantic connections between entities. Figure 25 summarizes the symbols in the ORM modeling language that we will use in this paper.

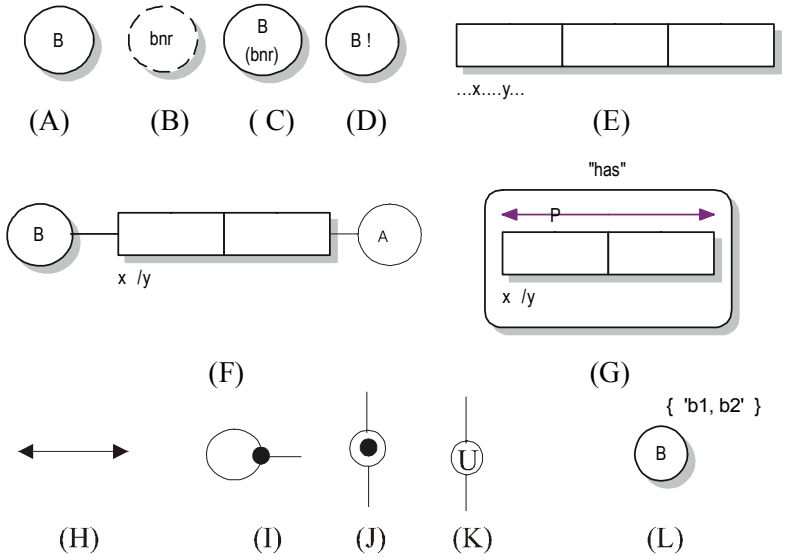


Fig. 25: Main symbols in Object-Role Modeling (ORM).

Atomic *entities* (figure 25A) or *data values* (figure 25B) are modeled in ORM as simple (hyphenated) circles. Instances of an entity type furthermore can exist independently (e.g. they are not enforced to participate in any relationship), which is shown by adding an exclamation point after the entity type's name (figure 25D). *Simple* reference schemes in ORM are abbreviated by putting the *value type* or *label type* in parenthesis beneath the name of the entity type (figure 25C). Semantic connections between entities are depicted as combinations of boxes (figure 25E) and are called *facts* or *fact types* in ORM. Each box represents a role and must be connected to either an *entity type*, a *value type* or *a nested object type* (see figure 25F). A fact type can consist of one or more roles. The number of roles in a fact type is called the fact type arity. The semantics of the fact type are put in the *fact predicate* (this is the text string ...*x*...*y*... in figure 25E). A *nested object type* (see figure 25G) is a non-atomic entity type that is connected to a fact type that specifies what the constituting entity types and/or values types are for the nested object type.

Figures 25H through 25L illustrate the diagramming conventions for a number of *static population constraint(s) (types)* in ORM. A double-arrowed line (figure 25H) that covers one or more 'boxes' of a fact type is the symbol for an *internal uniqueness constraint*. The symbol in figure 25K stands for an *external uniqueness constraint*. A(n) uniqueness constraint restricts the number of identical instances of a role combination 'under' the uniqueness constraint to *one*. A *mandatory role constraint* (figure 25I) can be added to a role. It specifies that each possible instance of such an object type must play that designated role at *all* times. A *disjunctive mandatory role constraint* (figure 25J) is defined on two or more roles and specifies that each possible instance of the object type connected to these roles must *at least* play *one* of these roles at *any* time. In figure 24L an example of a value constraint is given that enforces that each instance of the object type B either has the value *b1* or *b2*. An in-depth treatment of ORM can be found in [5].

APPENDIX B: UML CLASS DIAGRAM MODELING CONCEPTS

UML contains 9 types of diagrams: *class* diagrams, *object* diagrams, *use case* diagrams, *sequence* diagrams, *collaboration* diagrams, *state chart* diagrams, *activity* diagrams, *component* diagrams and *deployment* diagrams [1]. There is only one diagram type in UML that encodes the conceptual view of the static model on a type level: the *class* diagram.

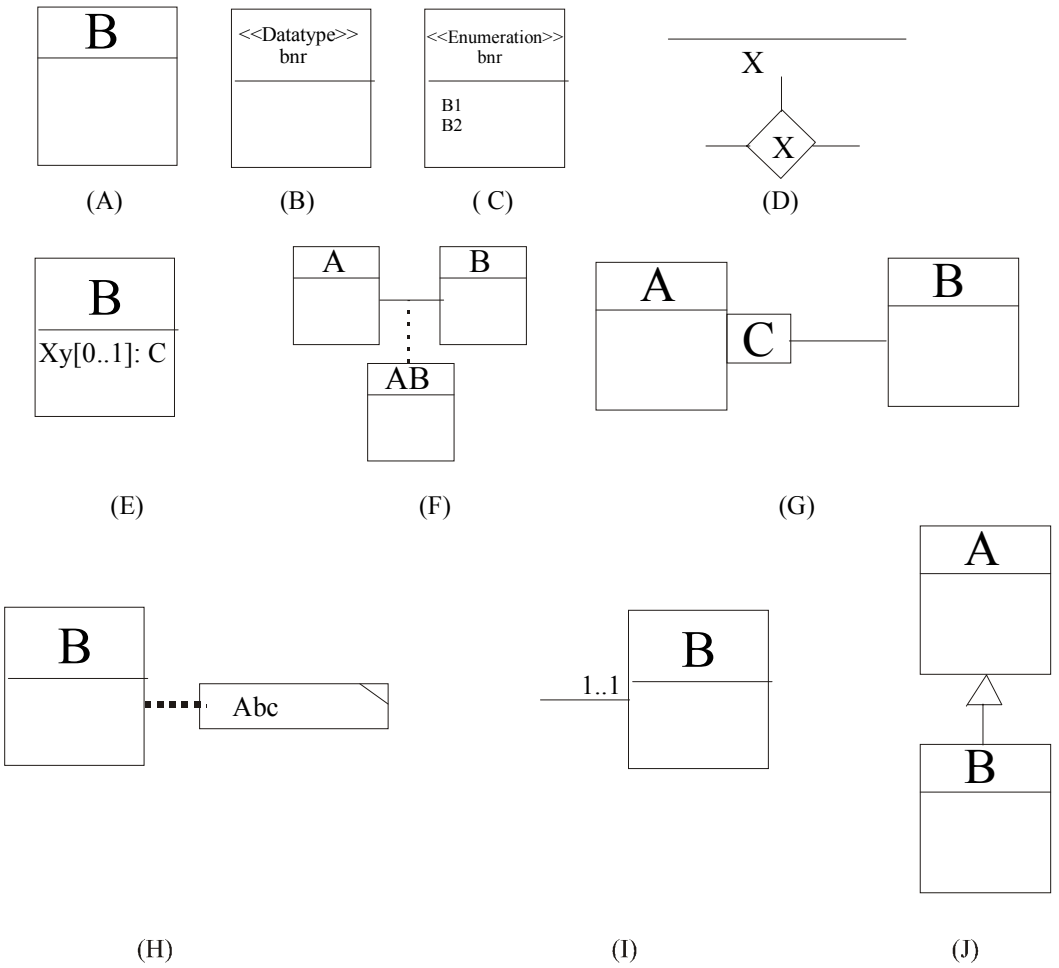


Fig. 26: Main symbols used in the static structure of the Unified Modeling Language class diagram

A class diagram additionally contains parts of the behavioural description for an application subject area: a list of *methods* that can be executed by a specific object class. In this article we will encode the static structure by using the relevant modeling constructs in an object class diagram (see figure 26). The main building block of a UML class diagram is an *object class* (figure 26A). The concepts of *data type* (figure 26B) and *enumeration data type* (figure 26C) are defined in UML as well. The latter modeling construct can only be applied when the possible instances of the data type are known. Semantic connections between instances of object classes in UML can be encoded either as an *association* (figure 26D) or as a *class attribute* (figure 65E).

Non-atomic application concepts can be encoded as *association classes* (figure 26F) and complex-naming conventions can be represented using a *qualifier attribute* in UML (figure 26G). Static constraints on the data structure can be represented in UML using the *enumeration data type* (figure 26C), the *textual constraint* (figure 26H) , the *attribute multiplicities* ([0..1] in figure 26E) and the *association end multiplicity* (figure 26I). Finally, in UML class diagrams the concept of subclass exists (in figure 26J B is a subclass of class A). An in-depth treatment of UML can be found in [1].

APPENDIX C: ORM-TO-UML TRANSFORMATION ALGORITHMS

C.1 Transformation algorithms for individual fact configurations

Algorithm 1: ORM-Unary-to-UML-class-attribute(ftx)
BEGIN
*Get the entity type that plays the role in the unary fact type **ftx** {A}*
*Get the reference scheme for the entity type **A** {anr}.*
*Get the predicate of the unary fact type **ftx** {xy}.*
*IF no object class exists with name **A***
*THEN Create a UML object class with name **A**.*
*Create the first class attribute for the object class **A** having name **anr**.*
*Add the qualification **{P}** to the attribute **anr**.*
ENDIF
*Create a class attribute for object class **A** that has the name **xy**.*
*Add the attribute type **boolean** to the class attribute **xy**.*
*IF entity type **A** does not play a mandatory role in fact type **ftx***
*THEN assign an attribute multiplicity of [0..1] to the attribute **xy***
ENDIF
END

Algorithm 2: ORM-Unary-to-UML-sub type(ftx)
BEGIN
*Get the entity type that plays the role in the unary fact type **ftx** {A}*
*Get the reference scheme for the entity type **A** {anr}.*
Get the predicate of the unary fact type {xy}.
*IF no object class exists with name **A***
THEN
*Create a UML object class with name **A**.*
*Create the first class attribute for the object class **A** having name **anr**.*
*Add the qualification **{P}** to the attribute **anr**.*
ENDIF
*Create a sub type of object class **A** that has the name **xy**.*
END

Algorithm 3: ORM-Binary-to-UML-attribute(ftx,A)[6]
BEGIN
*IF the object class with name **A** does NOT exist*
*THEN Create a UML object class with name **A***
*Get the reference scheme for the entity type **A** {anr}*
*Create the first class attribute for the object class having name **anr***
*Add the qualification **{P}** to the attribute **anr***
ENDIF
*Add an attribute to object class **A** which has as name the appropriate predicate of fact type **ftx {**ax}*
IF the object type in the other role {B} is a value type
*THEN assign this value type as the data (attribute) type of attribute **ax**.*
*ELSE append the name of object type **B** to the name of attribute **ax.***
*IF object type **B** has an abbreviated reference scheme*
THEN get the reference scheme for object type B {bnr}
*Assign **bnr** as the data (attribute) type of attribute **ax**.*
ENDIF
ENDIF

[6] We remark that this mapping can also be used for *nested object types* (see section 2.7)

*IF there exists a uniqueness constraint defined on the role of fact type played by object type **A***
*THEN assign a maximum attribute multiplicity to **ax** of **[..1]***
*ELSE assign a maximum attribute multiplicity to **ax** of **[..*]***
ENDIF
*IF there exists a uniqueness constraint defined on the role played by object type **B***
*THEN IF object type **A** plays a(n) implied mandatory role in **ftx** AND a uniqueness constraint defined on the role of fact type **ftx** played by object type A exists*
*THEN place {U} behind the attribute **ax***
*ELSE assign the following textual constraint to object class **A**: {each element of {ax} refers to at most one A}.*
ENDIF
ENDIF
IF there exists a uniqueness constraint defined on both roles
*THEN assign a maximum attribute multiplicity to **ax** of **[..*]***
ENDIF
*IF (entity type **A** is independent) OR (**A** does not play a(n) implied mandatory role in **ftx** in the global schema)*
*THEN add a lower multiplicity of **[0..** to attribute **ax***
*ELSE add a lower multiplicity of **[1..** to attribute ax*
ENDIF
IF the final attribute multiplicity for attribute ax = [1..1]
THEN leave out the multiplicity because it is the UML default
ENDIF
END

Algorithm 4: ORM-Binary-to-UML-association(ftx)
BEGIN
WHILE still roles in ftx
DO take next role {rx}
*get the object type that plays role **rx** {ex}*
*IF there does not exist a classifier for the object type **ex***
THEN
*IF **ex** is an entity type*
*THEN create a UML object class with name **ex***
*Get the reference scheme for the entity type **ex** {enr}*
*Create the first class attribute for the object class **ex** having name **enr***
*Add the qualification **{P}** to the attribute **enr***
*ELSE Create a UML data type with name **ex***
ENDIF
ENDIF
ENDWHILE
*Create a binary association (line) {bx} between the object class(es)[7] and or data type that play the roles in the binary fact type **ftx**.*
*Give the association **bx** a name that is equal to the appropriate predicate of **ftx***
*IF role names exist in the ORM fact type **ftx***
*THEN Match the role names in fact type **ftx** with the association end names in the association **bx**.*
ENDIF
*IF there is a uniqueness constraint defined on **ftx** that covers both roles*
*THEN assign an upper multiplicity of **..*]** to both association ends in the association **bx.***
ENDIF
WHILE still roles in ftx

[7] This can also be an association class

DO take next role {nrx}
IF there is a uniqueness constraint defined exclusively on role nrx
*THEN assign an upper multiplicity of **..1]** to the opposite association end in the corresponding association **bx**.*
*ELSE assign an upper multiplicity of **..*]** to the opposite association end in the corresponding association bx.*
ENDIF
*IF (a mandatory role is defined on role) **nrx** OR (**nrx** is the only role that is played by the object type and the object type is NOT independent}*
*THEN assign a lower multiplicity of **[1..]** to the opposite association end in the corresponding association **bx**.*
*ELSE assign a lower multiplicity of **[0..]** to the opposite association end in the corresponding association **bx**.*
ENDIF
IF the final association end multiplicity for the opposite association end = [1..1]
*THEN replace the association end multiplicity for the opposite association end by **1***
ENDIF
IF the final association end multiplicity for the opposite association end = [0..]*
*THEN replace the association end multiplicity for the opposite association end by *****
ENDIF
ENDWHILE
END

Algorithm 5: ORM-Nary-to-UML-association(ftx)
BEGIN
WHILE still roles in ftx
DO take next role {rx}
*Get the ORM object type that plays role **rx** {ex}*
*IF there does not exist a classifier for the object type **ex***
THEN
*IF **ex** is an entity type*
*THEN create a UML object class with name **ex***
*Get the reference scheme for the entity type **ex** {enr}*
*Create the first class attribute for the object class **ex** having name **enr***
*Add the qualification **{P}** to the attribute **enr***
*ELSE Create a UML data type with name **ex***
ENDIF
ENDIF
ENDWHILE
*Create a N-ary association {bx} between the object class(es) that play the roles in the N-ary fact type **ftx** by connecting the object classes one time for every role in **ftx** to the diamond in **bx**.*
*Give the association **bx** a name that is equal to the predicate of **ftx***
*IF role names exist in the ORM fact type **ftx***
*THEN Match the role names in fact type **ftx** with the association end names in the association **bx**.*
ENDIF
IF there is a uniqueness constraint defined on ftx that covers all roles
*THEN assign an upper multiplicity of **..*]** to all association ends in the association **bx**.*
ENDIF
*WHILE still roles in **ftx***
DO take next role {nrx}
IF there is a uniqueness constraint defined on the other [N-1] roles
*THEN assign an upper multiplicity of **..1]** to the corresponding association end **anrx** in the association **bx**.*

*ELSE assign an upper multiplicity of **..*]** to the corresponding association end **anrx** in the association **bx**.*
ENDIF
*IF a mandatory role is defined on role **nrx** OR role **nrx** is the only role that is played by the object type*
*THEN connect a textual constraint: {'Each **ex** must play this role'} to the line in the class diagram that connects the role to the association diamond*
*Assign a lower multiplicity of **[0..]** to the corresponding association end **{anrx}** in the association **bx.***
*ELSE assign a lower multiplicity of **[0..]** to the corresponding association end **{anrx}** in the association **bx**.*
ENDIF
*IF the final association end multiplicity for this **anrx** = [1..1]*
THEN replace the association end multiplicity for this role by 1
ENDIF
*IF the final association end multiplicity for this **anrx**= [0..*]*
*THEN replace the association end multiplicity for this role by **
ENDIF
ENDWHILE
END

Algorithm 6: ORM nested object type-onto-UML-association class (nt)
BEGIN
*Get the defining fact type for the nested object type **nt** {fnt}*
*Get the UML association for **fnt** {as}*
*Add an object class with name **nt***
*Add a hyphenated line from the line or diamond to object class **nt***
*Remove the name from the association **as***
END

Algorithm 7: co-referenced-object-type-onto-UML-attributes (D)
BEGIN
*Get the co-referenced object type **D***
IF no UML object class D exists
*THEN Create a UML object class with name **D***
ENDIF
WHILE still fact types left on which an external uniqueness (co-reference) constraint is defined
DO take next fact type {nfx}
IF object type that plays the role on which the co-reference constraint is defined is an entity type
THEN get this entity type {ex}
*get the abbreviated reference scheme for **ex** {anx}*
*add an attribute to the object class **D** with attribute name of the fact type predicate combined with the name of the entity type **ex** and as attribute type the name of the value type **anx** {enx}*
ELSE get the value type {anx}
*add an attribute to the qualifier with attribute the name of the fact type predicate and as attribute type the name of the value type **anx** {enx}*
ENDIF
*add the qualification {P} to the attribute **enx**.*
ENDWHILE
END

Algorithm 8: ORM subtype-onto-UML-subtype (SUBT)
BEGIN
*Get the ORM super type of **SUBT** {SUPERT}*
*IF there does not exist an object class for **SUPERT***
*THEN create a UML object class with name **SUPERT***
*IF an abbreviated reference scheme exists for **SUPERT***
*THEN Get the reference scheme for the entity type **SUPERT {snr}***
*Create the first class attribute for the object class having name **snr***
*Add the qualification {P} to the attribute **snr***
ENDIF
ENDIF
*IF there does not exist a UML object class for **SUBT***
*THEN Create a UML object class with name **SUBT***
ENDIF
*Create the UML subtype by drawing a subtype triangle under object class **SUPERT** and a connecting line between this super type triangle and **SUBT***
*IF a subtype defining rules exists for **SUBT***
THEN
For each entity type (sub or super type) in the subtype defining ORM rule draw a hyphenated line to the corresponding object class or attributes.
Connect all hyphenated lines to the sub typing defining rule in brackets. Copy the subtype-defining rule as a textual constraint attached to the hyphenated lines
ENDIF
END

C.2 Transformation algorithms for complete ORM conceptual schema to complete UML class diagram

Algorithm 9a: ORM-onto-UML-initial class diagram (ORM conceptual schema, unary encoding preference)
BEGIN
Get all entity types, nested object types and functional value types[8] in the ORM conceptual schema {entity types left}
Get all fact types in the ORM conceptual schema {fact types left}
*WHILE still entity types or nested object types in **entity types left***
*DO get next entity type or nested object type **{ex}***
*IF [(**ex** is an independent entity type OR **ex** plays two or more roles of which at least one role is mandatory)AND (**ex** has an abbreviated reference scheme)]*
*THEN get value type from abbreviated reference scheme **{vnrx}***
*model **ex** as an explicit object class **{ocx}** and model the value type **vnrx** of the abbreviated reference scheme as the attribute name of the identifier attribute **{ocanr}***
*add the qualification {P} after attribute **ocanr***
*Remove **ex** from **entity types left***
ELSE IF ex= nested object type
*THEN get constituting fact type for **ex** {ftx}*
*IF arity of constituting fact type **ftx** >2*
*THEN algorithm 5 (**ftx**, **lower multiplicity mode**)*
*ELSE algorithm 4 (**ftx**)*
ENDIF
*algorithm6(**ex**)*

[8] A *functional value type* is a *value type* that plays a role in a binary fact type which is no part of a co-reference scheme on which **no** *entity type* or *nested object type* is defined.

*remove **ex** from **entity types left***
*remove **ftx** from **fact types left***
*ELSE IF **ex** is a super type AND **ex** has an abbreviated reference scheme*
*THEN model **ex** as an explicit object class **{ocx}** and model the value*
*type **vnrx** of the abbreviated reference scheme as the attribute*
*name of the identifier attribute **{ocanr}***
*add the qualification {P} after the attribute **ocanr***
*remove **ex** from **entity types left***
*ELSE IF **ex** is a subtype*
*THEN algorithm8**(ex)***
*remove **ex** from **entity types left***
ENDIF
ENDIF
ENDIF
ENDIF
*IF (**ex** plays a role in at least one ternary or higher order fact type {FTY} AND **ex** is not a nested object type) OR*
*(**ex** plays two roles in at least one binary fact type{FTY}) OR*
*(**ex** plays a role in at least one unary fact type {FTY})*
*THEN add the fact types in FTY that are not defining fact types for a nested object type to the set of fact types to be created in this sub transformation {**naryset**}*
ENDIF
ENDWHILE
*WHILE still fact types in **naryset***
*DO get next fact type **{ftx}***
IF arity of ftx>2
THEN
*Algorithm 5(**ftx,lower multiplicity mode**)*
ENDIF
IF arity of ftx=1
THEN IF unary encoding preference=subtype
*THEN algorithm 2(**ftx**)*
*ELSE algorithm 1(**ftx**)*
ENDIF
ENDIF
IF arity of ftx=2
*THEN Algorithm 4(**ftx**)*
ENDIF
*Remove fact type **ftx** from **fact types left***
ENDWHILE
WHILE still value types left
DO get next value type{vx}
*IF a value constraint is defined on the value type **vx***
*THEN create an enumeration **vx** for that value type and list the values in the value type as enumeration constants of the enumeration **vx***
ENDIF
ENDWHILE
END

Algorithm 9b: UML initial class diagram-onto-UML-second class diagram (ORM conceptual schema, UML class diagram, entity types left, fact types left)
BEGIN
*WHILE still object types in **entity types left***
*DO get next entity type **{ex}***
IF [(ex plays one role in one binary fact type) AND (ex is not independent) AND (ex has an abbreviated reference scheme OR ex is a functional value type)]AND (the other role in that fact type in which ex plays the role is played by an entity type that is already encoded as an object class that is not an association class).
*THEN Get the other entity type **{oex}***
*get fact type predicate **{fty}***
*IF **fty** is in **fact types left***
THEN algorithm3(fty,oex)
*remove **fty** from **fact types left***
ENDIF
*remove **ex** from **entity types left***
*ELSE IF **ex** is a co-referenced entity type*
THEN IF at least one of the entity types in the functional roles of the constituting fact types in the co-reference is modeled as an explicit object class
THEN apply the nest/co-reference theorem.
Get the defining fact type of the nested object type {ftx}
*IF arity of **ftx**>2*
THEN Algorithm 5(ftx,lower multiplicity mode)
ELSE Algorithm 4(ftx)
ENDIF
Algorithm 6(ex)
*Remove **ex** from **entity types left***
*ELSE Algorithm 7 **(ex)***
*Remove **ex** from **entity types left***
ENDIF
ENDIF
ENDIF
ENDWHILE
END

Algorithm 9c: UML second class diagram –onto-UML third class diagram (ORM conceptual schema, UML class diagram, entity types left, fact types left)
BEGIN
*WHILE still object types in **entity types left***
DO get next object type {ex}
*IF **ex** plays one role each in only binary fact type(s) AND no mandatory role is defined on those roles AND at least one of the other entity types in the binary fact type(s) are encoded as object classes*
*THEN get name from the abbreviated reference scheme for **ex** {vnx}*
*Get the binary fact types in which **ex** plays a role {fcx}*
*WHILE still fact types in **fcx***
Get next fact type {fxy}
DO get the object type in the other role {ocx}
*Get the fact type predicate of **fxy***
*IF (**ocx** is a nested object type AND the configuration of **fxy**=2.4, 2.6 or 2.8) OR (**ocx** is an entity type AND the configuration of **fxy**=2.1, 2.2, 2.4, 2.5, 2.6, 2.8 or 2.9)*
*THEN Algorithm 3 **(fxy, ocx)***
*Remove **fxy** from **fact types left***
*ELSE IF there does not exist a classifier for the object type **ex***
THEN

*IF **ex** is an entity type*
*THEN create a UML object class with name **ex***
*Get the reference scheme for the entity type **ex** {enr}*
*Create the first class attribute for the object class **ex** having*
*name **enr***
*Add the qualification **{P}** to the attribute **enr***
*ELSE Create a UML data type with name **ex***
ENDIF
ENDIF
ENDIF
ENDWHILE
ENDIF
ENDWHILE
END

Algorithm 9d: UML third class diagram-onto-UML-fourth class diagram (ORM conceptual schema, UML class diagram, entity types left, fact types left)
BEGIN
*Get the remaining object types that are no value types from **entity types left** that are not yet encoded as an object class {REST}*
WHILE still entity types in REST
DO take next object type {ex}
*Calculate the number of roles in **fact types left** in which **ex** is involved {nex} AND in which the other object type is not yet encoded as an object class or association class*
*IF **nex**=0 THEN remove **ex** from **entity types left** ENDIF*
ENDWHILE
Rank the entity types in REST according to the values for nex. Make a list with the entity type that has the highest next on top and so forth until the entity type with the lowest nex is on the bottom of the list {LIST}
WHILE still entity types in LIST
DO take next highest entity type in LIST (>0) {nel}
*WHILE still binary fact types in which **nel** is involved*
*DO take next fact type ein which **nel** is involved from **fact types left**{fnel}*
*Get the other object type in **fnel {oel}***
*Get the fact type predicate of **fnel {fny}***
*Algorithm 3 **(fny, nel)***
*Remove **fny** from **fact type list***
*Subtract 1 from the value for **oel** in the list*
ENDWHILE
*Remove **ne**l from LIST and from **entity types left***
ENDWHILE
END

Algorithm 9e: UML fourth class diagram-onto-Final UML model (ORM conceptual schema, UML class diagram, unary encoding, fact types left)
BEGIN
*WHILE still fact types in **fact types left***
*DO take next fact type {**ftx**}*
*IF arity **ftx**=2*
THEN
Algorithm 4 (ftx)
*Remove **ftx** from **fact types left***
ENDIF
ENDWHILE
Get all entity types/nested object types in the ORM model {OBTYP}
*WHILE still entity types/nested object types in **OBTYP***
*DO take next entity type/nested object type {**eox**}*
*IF[(**eox** is encoded as an object class and is NOT independent) AND (**eox** does not play a mandatory role in the ORM conceptual schema) AND(eox plays at least two roles)] OR [On a subset of the roles that eox plays a disjunctive mandatory role constraint is defined]*
*THEN assign the following textual constraint to the object class eox and all associations and attributes that are played by **eox** or on which the disjunctive mandatory role constraint is defined: {every instance of object class eox should participate in at least one of the associations} OR {eox.attr1 is not null or ….or eox.attrN is not null}*[9]
ENDIF
ENDWHILE
END

[9] See for an example of this constraint Halpin [5, p. 354, figure 9.3]